

LARGE DATABASE SEARCH USING CAM AND HASH

TECHNICAL FIELD

5 The present invention relates generally to static information storage and retrieval systems, and more particularly to architectures for searching large databases of stored data using hash algorithms or associative memory, also referred to as content addressable memory or tag memory.

BACKGROUND ART

10 Many real world systems require searching information at very high speed; hence hardware based approaches are often employed. An increasingly common example, and the one which will primarily be used herein, is searching in network systems. In information searching there are basically two types of information searching: exact match and partial match searching. 15 The present invention is intended to solve problems in exact match searching. Two approaches widely used today for exact match searching are hash-based searching and content addressable memory or CAM-based searching.

20 To compare these two types, for instance, one can consider the problem of designing and searching a database that can support one million entries of 128-bit input data and 32-bit of associated data.

25 In a hash based search, the concept is to map the search data, which is often very lengthy as in this 128-bit example, to numbers which are often much smaller, say only 20-bits in length, since only 20 bits are needed to address one million entries. Since this conversion (128-bit to 20-bit) is not a one-to-one mapping, collisions will very likely occur. That is, a hash collision is said to have occurred here when the hash function employed returns the same 20-bit result for two different 128-bit inputs.

30 Furthermore, depending on the particular data encountered and the particular hash function employed, more than just two different 128-bit inputs can be mapped to the same 20-bit result number. It therefore is not uncommon for a system to have to be designed to accommodate the fact that three, four or even more different inputs will be mapped by the hash function to the

very same output number. There are many different approaches to reducing the impact of hash collisions. One approach is to increase the database size. For instance, consider first the degenerate case, a "1-way-set-associative hash." Here the 128-bit input search values are mapped to 20-bit values used as address indexes into a memory. The memory needs to be 1M in size, since each 20-bit value needs to map to an actual memory location. This case, of course, does not handle any hash collisions at all. It simply ignores them, and therefore is not very practical. Consider next a "2-way-set-associative hash." Here one set of collisions can be handled. The memory needs to be 2M in size, since each 20-bit value needs to map to two actual memory locations. Higher-way associative hashes can also be used, but beyond a "4-way-set-associative hash" this approach of simply increasing the database size is typically not practical. Based on the parameters used here, in a 4-way case the memory would need to be 4M in size and would be very poorly utilized.

FIG. 1 (background art) is a block diagram depicting a search engine 10 using conventional hash-based database searching. A controller 12 includes a hash function 14 which can receive 128-bit input search values and generate 20-bit hash value which is used as an index to address a memory 16.

If we assume that hash collisions will not happen too often, and use only a 2-way-set-associative hash, the memory 16 needs to be able to store a database having two million entries. This is depicted here as a base region 16a and a conflicts region 16b in the memory 16, a total of two regions for a 2-way-set-associative hash. A 21-bit wide address bus 18 is therefore needed to communicate between the controller 12 and the memory 16 (20-bits to address the first one million entries in the base region 16a, and one additional bit to address the second million entries, in the conflicts region 16b used to support the one set of potential collision cases). The entries in the memory 16 each require 160 bits, 32 bits for an associate value which is the desired result of the search and 128 bits for the input search value which is intended to produce that particular result.

The above illustrates a key point in hash-based database searching -- both the associate content value and a stored instance of the search value which produces it must be stored in the memory 16 and returned to the controller 12, here via a 160-bit result bus 20, for the controller 12 to perform a comparison to determine if a hash collision has occurred. The controller 12 can only do this if it has both the input and stored search values. If these are the same in a search

result obtained from the base region 16a, the associate value in the search result is valid and the controller 12 can carry on with using it. If these values are different, however, a hash collision has occurred. Then the controller 12 accesses the memory 16 a second time, using the 21st bit to address an entry stored in the conflicts region 16b. If the input and stored search values are now the same, the controller 12 can again carry on. If these are still different, however, another hash collision has occurred and using only a 2-way-set-associative hash and a two million entry database is not adequate for the task at hand.

A good hash algorithm is one that produces very few collisions. However, it usually cannot be known how many collisions will actually occur, because the pattern of the input search data is typically not known in advance. If there is more than one collision for a given number, a 2-way-set-associative hash will not be able to handle it. In order to support the database with more confidence, 4-way or more set-associative approach should then be used. When such is used, more memory must be provided.

The size of the memory depends on two things: the number of entries being supported and the number of ways of set-associativity employed. For example, to support n entries using 4-way-set-associativity, the memory size has to have 4n entries. For one million entries this means that the memory must be four million entries "deep," even though only one million of entries will ever potentially be used, i.e., less than 25% of the memory is utilized.

The number of ways of set-associativity also dictates that more clock cycles will potentially be needed for a search. As noted for the 2-way associative hash in FIG. 1, it will take a maximum of two memory read operations (instead of one) to perform a database search, since one collision may happen during the search. Similarly, for an m-way set-associativity it may take up to m memory read operations to perform a database search.

It follows that hash-based database searching has substantial disadvantages when one considers the large amount of memory needed and the limited speed of searching possible. For discussion, these can be termed the memory size issue and the search speed issue, and both increase linearly when hash-based searching is used. The memory size issue is largely a matter of cost, and thus may be easily solved if cost is not a serious problem. The search speed issue is more difficult, however, and it may make hash-based searching impractical in many applications.

FIG. 2 (background art) is a basic block diagram depicting a search engine 50 using conventional CAM-based database searching. Here a controller 52 provides a 128-bit input

search value to a CAM 54 that will search its own database (of one million entries) and provide a 20-bit index value to a memory 56, where up to one million 32-bit associate content values may be stored.

Although other types of memory can theoretically be used in place of the CAM 54, content addressable memory, or associative memory as it is also widely known, is particularly useful. When provided an input search value, a content addressable memory will very rapidly provide as its output the address within it of any match (multiple matches should not occur here unless the CAM 54 is improperly programmed). This index, perhaps with appropriate translation by a suitable logic circuit, can then be used as an address into the memory 56.

The controller 52 here provides the input search value to the CAM 54 via a 128-bit wide search data bus 58; the CAM 54 provides the address index value to the memory 56 via a 20-bit wide address bus 60, and the memory 56 provides the search result to the controller 52 via a 32-bit wide result bus 62. Since the CAM 54 always provides, if anything, a unique address in the memory 56, only one read operation is required.

TABLE 1 summarizes, along with aspects of the present invention which are discussed presently, the differences between the prior art hash-based and CAM-based approaches when the controllers 12, 52 are ASIC devices and the memories 16, 56 are RAM. From this a number of respective advantages and disadvantages for these approaches become readily apparent. For instance, the hash-based approach has lower cost and power consumption. The CAM-based approach provides a lower pin count at the ASIC (which is highly desirable), uses less memory (overall), never encounters collisions (no dependency is necessary on the nature of the input data values or on the choice of an algorithm), and has a potentially higher search speed (one which is consistent and known). There are still other advantages and disadvantages, but these are the typical major ones.

Accordingly, it is desirable to find hardware-based approaches of database searching which do not suffer from the respective disadvantages of the conventional hash-based and CAM-based approaches, and which retain or provide additional advantages for these prior art approaches.

DISCLOSURE OF INVENTION

Accordingly, it is an object of the present invention to provide components to facilitate the construction of more efficient database search engines.

5 Another object of the invention is to provide components to facilitate the construction of database search engines which employ hash-based techniques yet are less restricted by hash collisions.

10 And another object of the invention is to provide configurable and cascade able components to facilitate the construction of more adaptable and powerful database search engines.

11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
220

provided and create an address value based on that, which it provides on an output bus connecting the circuit to the external memory. The address value represents an address in the external memory. The logic unit also receives the hash output and, if no CAM output is provided, it creates a hash address based on that. The logic unit may also receive a hash pointer value and creates the address value based on that. The logic unit yet further receive a hash hit signal, and responsive to that provide the address value on the output bus. A search data storage stores a number of the hash pointer values, representing potential instances of the hash addresses. The search data storage also stores a number of search data values, representing potential instances of the search values. When the search data storage receives a hash address from the logic unit it retrieves a hash pointer value which it provides to the logic unit. The search data storage also retrieves a search data value based on the hash pointer value. An embedded comparator is connected to the input bus and able to receive the search value. It also receives the search data value from the search data storage, and compares the search value and the search data value to determine whether a match exists. If so, it provides the hash hit signal to the logic unit.

An advantage of the present invention is that it provides a direct cost savings due to improved memory utilization, and it provides indirect cost savings due to reduced bus widths, reduced pin counts, reduced circuit and component footprints, and reduced power usage and heat dissipation needs.

Another advantage of the invention is that it is easily implemented within many existing search engine schemes, thus permitting rapid adoption and minimal industry change.

Another advantage of the invention is that it is essentially plug-in usable in many existing search engine schemes.

And another advantage of the invention is that its speed equals or betters hash-based prior art, which suffers from poor utilization and limited hash collision handling ability, and the speed of the invention approaches that of the CAM-based prior art, which is expensive.

These and other objects and advantages of the present invention will become clear to those skilled in the art in view of the description of the best presently known mode of carrying out the invention and the industrial applicability of the preferred embodiment as described herein and as illustrated in the several figures of the drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

The purposes and advantages of the present invention will be apparent from the following detailed description in conjunction with the appended tables and figures of drawings in which:

5 TABLE 1 summarizes differences between the prior art hash-based and CAM-based approaches and a variation of the embodiment of the present invention depicted in FIG. 8;

 TABLE 2 summarizes inputs and stored values in an example described for the embodiment of the present invention depicted in FIG. 6 and the process for that embodiment depicted in FIG. 7; and

10 TABLE 3 provides a comparison of two different implementations of a one million entry 8-way-set-associative hash-based system, wherein one uses a typical prior art hash-based approach and the other uses the H-CAM with the indirect hash pointer and a cascading mechanism of the present invention.

 FIG. 1 (background art) is a block diagram depicting a simple hash-based database searching example;

 FIG. 2 (background art) is a block diagram depicting a simple CAM-based database searching example;

 FIG. 3 is a block diagram depicting a search engine according to a hash pointer based embodiment of the present invention;

20 FIG. 4 is a block diagram depicting a search engine according to a single-level hash-CAM based embodiment according to the invention;

 FIG. 5 is a block diagram depicting a search engine according to a quad-level hash-CAM based embodiment of the invention;

25 FIG. 6 is a block diagram depicting a search engine used in an example of a multi-leveled architecture embodiment of the invention, simplified for use in a detailed example;

 FIG. 7 is a flow chart depicting a process suitable for use with the H-CAM element of the embodiment of the invention shown in FIG. 6;

 FIG. 8 is a block diagram depicting a search engine according to an alternate embodiment of the invention, including a H-CAM element having a comparison section;

30 FIG. 9a-b show possible sub-architectures for parts of the H-CAM element of the embodiment shown in FIG. 8, wherein FIG. 9a is a block diagram depicting an architecture for

first a logic unit and FIG. 9b is a block diagram depicting an architecture for a search data storage;

FIG. 10 is a block diagram depicting a search engine according to an alternate embodiment of the invention, including a H-RAM element;

5 FIG. 11a-b show possible sub-architectures for parts of the H-RAM element of the embodiment shown in FIG. 10, wherein FIG. 11a is a block diagram depicting an architecture for a first logic unit and FIG. 11b is a block diagram depicting an architecture for a search data storage;

10 FIG. 12 is a block diagram depicting a search engine incorporating many of the principals discussed for other embodiments;

FIG. 13 is a block diagram depicting a search engine having the ability to scale for use in large systems by cascading multiple H-CAM elements;

FIG. 14 is a block diagram depicting a search engine having the ability to scale for use in large systems by cascading multiple H-RAM elements; and

5 FIG. 15 is a block diagram depicting a search engine having the ability to scale for use in large systems by cascading multiple CAM, H-CAM and H-RAM elements.

106080" 6552660

BEST MODE FOR CARRYING OUT THE INVENTION

A preferred embodiment of the present invention is a system for large database searches which marries both hash-based and CAM-based advantages without the respective disadvantages. As illustrated in the various drawings herein, and particularly in the views of FIG. 3-6, 8, 10, and 12-15, preferred embodiments of the invention are depicted by the general reference characters **100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, and 1100.**

A SEARCH ENGINE USING A HASH POINTER:

FIG. 3 is a block diagram depicting a search engine **100** according to one preferred embodiment of the present invention. The particularly novel aspect in this embodiment is the use of a hash pointer unit **110**, which is interposed between a controller **112** and a memory **114**. For an example here, lets say that the controller **112** uses 128-bit wide input search values to search a database of up to one million entries of 32-bit associate content value stored in the memory **114**. The controller **112** includes a hash function **116** that generates hash addresses (which potentially may be hash collisions). The controller **112** may be substantially the same as the conventional controller **12** discussed with FIG. 1 (background art). Unlike the scheme used in FIG. 1, however, lets use 4-way set-associativity here. Accordingly, using four way access to up to one million addresses, a 22-bit address bus **118** is needed to connect the controller **112** to the hash pointer unit **110**.

The hash pointer unit **110** includes a 4M x 20 memory which may contain programmable unique "pointers" to the one million potential entries in the memory **114**. The hash pointer unit **110** is 4M deep to handle 4-way associativity to one million potential entries, but it need be only 20-bits wide to provide pointers able to access the actual one million potential entries in the memory **114**. A 20-bit wide pointer bus **120** can therefore be used to connect the hash pointer unit **110** to the memory **114**.

Since hash collisions are possible and must still be checked for, the memory **114** accordingly needs to include entries 160-bits wide (128+32). Individual such entries, or search results are then provided to the controller **112** via a 160-bit wide result bus **122**.

The operation here is similar to that of the prior art search engine **10** in FIG. 1, except

that the stored search values and associate content values are accessed "indirectly" through the pointer values stored in the hash pointer unit 110. A 128-bit input search value is used by the hash function 116 to generate a 20-bit hash output, which the controller 112 combines with an offset value to create a 22-bit hash address.

5 Any of many different schemes will work to convert the hash value to the hash address. For instance, the controller 112 may do this by prefixing a 2-bit offset value to the hash value. Thus, a "00" prefix can represent that the hash output maps to the first "way" in the 4-way set-association used here, a "01" can represent the second way, and so fourth. On a first try with a particular 128-bit input search value, "00" may be prefixed to the hash value. If the search result
10 this produces has a stored search value that matches the original input search value, there has been no hash collision. In contrast, if these values do not match, a hash collision has occurred. Then "01" may be prefixed to the (same) hash value, creating a new hash address, and the hash pointer produced by the hash pointer unit 110 in response to this may similarly be used to retrieve a different search result. This can continue, handling up to four 128-bit input search
15 values that the hash function maps to the same hash value in this embodiment.

Summarizing, the search speed issue (as termed in the Background Art section) is not changed. However, the memory size issue is greatly changed. Whereas a 4M x 160 memory would be required using a prior art 4-way set-associative hash-based approach, the inventive search engine 100 here requires only a 1M x 160 memory and a 4M x 20 memory -- a substantial
20 overall savings.

A SINGLE LEVEL H-CAM AND A SEARCH ENGINE EMPLOYING IT:

FIG. 4 is a block diagram depicting a search engine 200 according to another preferred
25 embodiment of the present invention. The particularly novel aspect of this embodiment is a single level H-CAM 210. The single level nature of the H-CAM 210 should not be confused with the set-associative nature of a hash approach. That is, the "level" and "way" are different concepts. The H-CAM 210 in the example here is 1-way-set-associative, but that is coincidental with the use here of only a single level.

30 The search engine 200 further includes a controller 212 and a memory 214, which both may be substantially conventional. For instance, the controller 212 may be an application

specific integrated circuit (ASIC) and the memory 214 may be an external random access memory (RAM).

The controller 212 communicates an input search value to the H-CAM 210, via a search data bus 216, and the H-CAM 210 then communicates an address value to the memory 214, via an address bus 218. Responsive to the address value, a result value is communicated back from the memory 214 to the controller 212, via a result bus 220.

As shown, the architecture of the H-CAM 210 includes a hash unit 222, a CAM unit 224, and a logic unit 226. The input search value is provided, via a first path 228a, to both the hash unit 222 and the CAM unit 224. The hash unit 222 then provides a hash output on a second path 228b, and the CAM unit 224 may (if it finds a match) provide a CAM output on a third path 228c (the address within it of such a match). The logic unit 226 receives the CAM and hash outputs, via the paths 228b-c, and selects one for use as the address value, which it then provides to the memory 214 via the address bus 218 (effectively a last path 228d).

The following is an example pertaining particularly to the search engine 200 depicted in FIG. 4. Lets assume here that the search engine 200 will also use 128-bit wide input search values, to search a database of up to one million entries of 32-bit associate values. The search data bus 216 is 128 bits wide, the address bus 218 is 21 bits wide, and the result bus 220 is 160 bits wide.

The hash unit 222 employs a hash algorithm to map each 128-bit wide input search value to a "narrow" 20-bit wide hash output value. Each input search value received by the hash unit 222 causes it to provide a respective hash output to the logic unit 226. Of course, the hash output will depend on the hash algorithm actually used.

The CAM unit 224 in this example is a 100K x 128 content addressable memory. If an input search value is received in the CAM unit 224 that matches a stored comparison value (a 128-bit wide copy of a previous search value that caused a hash collision), the address of that match is provided by the CAM unit 224, as a CAM output, to the logic unit 226. In contrast, however, if an input search value is received which does not match any comparison value, no CAM output is provided.

The logic unit 226 here operates in a straight forward manner. A CAM output has higher priority than a hash output. If the logic unit 226 receives only a hash output, that becomes the basis of the address value. If it receives both a hash output and a CAM output, the CAM output

becomes the basis of the address value. This is so because the presence of a CAM output indicates that there is an already known and cataloged hash collision case. Using the hash output for the address value here would cause that collision to occur. Alternately, using the CAM output for the address value instead will avoid that collision. If the logic unit **226** receives only a CAM
5 output it can go ahead and use that as the basis for the address value. However, it may be desirable to treat this as an error condition in many embodiments, since a proper hash algorithm should always produce a hash output.

If necessary, the logic unit **226** also translates the hash or CAM output into an address value that appropriately maps to the memory **214**. For instance, the hash output may (naturally or
10 by translation) map to the low-order one million addresses in the memory **214**. If the CAM output also maps to the low-order one-hundred thousand addresses in the memory, it needs to be translated to instead map to high-order addresses in the memory **214**.

The memory **214** here includes a base region **214a** for storing search results that are accessed based on the hash outputs, and a conflicts region **214b** for storing search results that are
5 accessed based on the CAM outputs. The search results in the base region **214a** need to include a 128-bit wide stored search value and a 32-bit associate value, for reasons discussed presently. It then follows, because every possible hash output (defined by a 20-bit index value) must map to a search result, that the base region **214a** in the example here needs to have a size of at least 1M x 160.

The search results stored in the conflicts region **214b** of the memory **214**, however, do not have to include any stored search values. The CAM unit **224** already includes full 128-bit
20 wide copies of these same search values it uses to index into the conflicts region **214b**, and storing any stored search values in the conflicts region **214b** is unnecessary and wasteful. The conflicts region **214b** here can therefore simply contain the 32-bit associate values, and thus need
25 only be 100K x 32 in size.

The address bus **218**, memory **214**, and result bus **220** are stylistically shown in FIG. 4 in a manner to emphasize that the base region **214a** and the conflicts region **214b** may be different
in size, and that they may even be distinct physical devices. The result bus **220** needs to be 160
30 bits wide to carry either size of result value, but a 128-bit field can be empty whenever a result value is taken from the conflicts region **214b**.

The controller **212** receives a result value from the memory **214**, and it may check for a

new hash collision. The controller **212** can easily determine if the result value includes a stored search value. If not, the result value will have come from the conflicts region **214b** and the 32-bit associate value in the present result value can be used with confidence that it is uniquely associated with the original input search value. In contrast, for a non-empty stored search value, the result value will have come from the base region **214a** in the memory **214** and a hash collision is still a possibility that needs to be dealt with. The controller **212** can then compare the non-empty stored search value with the original input search value. If these are the same, no hash collision has occurred and the search engine **200** can simply carry on and use the 32-bit associate value in the present result value with confidence. However, if these values differ, a new hash collision has occurred and the associate value received as part of the result value should not be used. When a new hash collision is detected in this manner the search engine **200** specifically the CAM unit **224** and the conflicts region **214b** of the memory **214**, may be programmed to avoid it in the future. Theoretically, the search engine **200** here can handle hash collisions up to 100,001 deep in a worst case situation, since the conflicts region **214b** has 100K locations to map collisions to after a first entry in the base region **214a** has been used.

A MULTI-LEVEL H-CAM AND A SEARCH ENGINE EMPLOYING IT:

FIG. 5 is a block diagram depicting a search engine **300** according to another preferred embodiment of the present invention. The particularly novel aspect in this embodiment is a quad-level H-CAM **310**. A controller **312** and a memory **314** are also provided.

In a manner much similar to the previous embodiment, the controller **312** here communicates an input search value to the H-CAM **310**, via a search data bus **316**, and the H-CAM **310** then communicates an address value to the memory **314**, via an address bus **318**. Responsive to the address value, a search result is then communicated back from the memory **314** to the controller **312** via a result bus **320**.

As shown, the architecture of the H-CAM **310** includes a number of paired hash units (hash units **322a-d**) and CAM units (CAM units **324a-d**), and a single logic unit **326**. The input search value is provided to the hash unit **322a** and its paired CAM unit **324a**, via a path **328a**. The hash unit **322a** then provides a first hash output on a path **328b**, which conveys it to the next hash unit **322a** where a second hash output is provided on a path **328c**, and so forth until a last,

fourth hash output is provided on a path **328e**. Concurrently, the CAM unit **324a** provides a first CAM output on a path **328i**, a second CAM output is provided on a path **328h**, and so forth until a last, fourth CAM output is provided on a path **328f**. Summarizing, at this point there is a single final hash output on a path **328e** and multiple CAM outputs on paths **328f-i**, and all of these

5 paths **328e-i** lead to the logic unit **326**. The logic unit **326** selects a value from those it receives to use as the address value, and provides it to the memory **314** via the address bus **318**.

The following is an example particularly pertaining to the search engine **300** depicted in FIG. 5. Inside the H-CAM **310**, the path **328a** is 128 bits wide, as it is essentially the same as the search data bus **316**; the path **328b** is desirably smaller, say, 64 bits wide; the path **328c** is

10 desirably still smaller, say, 48 bits wide; the path **328d** is desirably yet smaller, say, 32 bits wide; and the paths **328e-i** are all smaller, specifically all 20 bits wide, as one of their contents is ultimately sent via the 21-bit wide address bus **318** (effectively a last path **328j**).

The purpose of the multiple hash units **322a-d** here is to reduce the input data space, hence the hash outputs should each desirably be smaller than the prior input. The multiple CAM

15 units **324a-d** store entries known to cause hash collisions at the respectively paired hash units **322a-d**. The underlying rationale of this multi-level architecture is that when a given number of hash collisions are expected a corresponding number of CAM entries will be needed. So, based on probability and knowing that the width of the CAMs shrink at successive levels, using multiple levels of CAM in this manner should reduce the total amount of CAM require to

20 provide for the expected number of hash collisions. Nevertheless, the actual number of hash collisions still depends on the particular pattern of input data encountered.

The logic unit **326** outputs an address value based on a highest priority scheme. The earlier CAM stages have higher priority over later stages, and ultimately over the hash output. For example, the CAM output of the first CAM unit **324a** has higher priority than the CAM

25 output of the second CAM unit **324b**, and so forth, ending here with the CAM output of the fourth CAM unit **324d** having priority over the hash output of the fourth hash unit **322d**. The logic unit **326** may also, if necessary, translate the outputs to an address value that appropriately maps to the memory **314**.

The memory **314** has a base region **314a**, a first conflicts region **314b**, a second conflicts

30 region **314c**, a third conflicts region **314d**, and a fourth conflicts region **314e**. As was discussed for the "highest" conflicts region **214b** in FIG. 4, the fourth conflicts region **314e** does not need

to include room for stored search values, since the first CAM unit **324a** contains a copy of the search value and will always match uniquely.

The hash output of the fourth hash unit **322d** becomes the basis for address values that map to the base region **314a**; the CAM output of the fourth CAM unit **324d** becomes the basis for address index values that map to the first conflicts region **314b**; the CAM output of the third CAM unit **324c** becomes the basis for address index values that map to the second conflicts region **314c**; and so forth. The conflicts regions **314b-e** thus each store data to handle correspondingly higher level collision cases.

A DIFFERENT MULTI-LEVEL H-CAM AND SEARCH ENGINE EMPLOYING IT, IN A DETAILED EXAMPLE:

FIG. 6 is a block diagram depicting a search engine **400** according to the present invention, now used in detailed example. A two-level H-CAM **410** is provided, along with a controller **412**, memory **414**, search data bus **416**, address bus **418**, and result bus **420**.

A programming unit **430** is also shown here (and many embodiments of the invention will include programming capability, but is not described in particular detail since its task can be performed in various ways and using various architectures that are all substantially conventional or which should be readily apparent to those skilled in the art in view of the following discussion. The result bus **420** also differs from the result bus **220** (FIG. 4) in that it is bi-directional, and the memory **414** here accordingly accommodates that. Some variations of the programming unit **430** may employ these when programming the memory **414**.

As shown, the architecture of the H-CAM **410** here includes a two hash units (hash units **422a-b**) and two CAM units (CAM units **424a-b**), and a logic unit **426**. A path **428a** connects the search data bus **416** to the first hash unit **422a** and the first CAM unit **424a**; a path **428b** connects the first hash unit **422a** to the second hash unit **422b** and the second CAM unit **424b**; a path **428c** connects the second hash unit **422b** to the logic unit **426**; a path **428d** connects the second CAM unit **424b** to the logic unit **426**; and a path **428e** connects the first CAM unit **424a** to the logic unit **426**.

The memory **414** includes a base region **414a**, a first conflicts region **414b**, and a second conflicts region **414c**. These might, for instance, respectively be 1M x 160, 200K x 160, and

100K x 32 in size.

FIG. 7 is a flow chart depicting a process **500** suitable for use with the H-CAM **410** of FIG. 6. The process **500** starts with a step **510**, where the two CAM units **424a-b** and the memory **414** are empty. In a step **512** an input search value is provided by the controller **412** to the H-CAM **410**. In a step **514** the input search value is received by and processed by the first hash unit **422a** to generate a first hash output. In a step **516**, the input search value is also received by and processed by the first CAM unit **424a** to create a first CAM output (but only if there is a match). In a step **518** the first hash output is received by and processed by the second hash unit **422b** to generate a second hash output. In a step **520**, the first hash output is also received by and processed by the second CAM unit **424b** to create a second CAM output (but, again, only if there is a match).

Summarizing, up to here the H-CAM **410** has internally produced a final (second) hash output, and possibly also either or both of first and second CAM outputs. Next, in a step **522** these outputs are received by and prioritized by the logic unit **426**. The first CAM output receives the highest priority; if it is present it is used for the address value. The second CAM output receives the next priority; if it is present and the first CAM output is not, the second CAM output is used for the address value. The (final) second hash output receives the lowest priority; if it is present and the neither of the CAM outputs is, the second hash output is used for the address value. If necessary, the logic unit **426** also translates the respective output used so that the address value maps to a corresponding area in the memory **414**.

In a step **524** it is determined whether a hit was detected in the first CAM unit **424a**. If so, in a step **526** the first CAM output is used to access the memory **414** and the controller **412** will receive a search result (typically only an associate value) stored in the second conflicts region **414c**. Since the contents of the second conflicts region **414c** are always uniquely associated with possible input search values, the process **500** is essentially finished for the present input search value, and the search engine **400** can carry on with whatever further operations on the search result are desired.

If a hit was not detected in the first CAM unit **424a**, in a step **528** it is determined whether a hit was detected in the second CAM unit **424b**. If so, in a step **530** the first CAM output is used to access the memory **414** and the controller **412** receives a search result stored in the first conflicts region **414b**.

There still remains a possibility, however, that the first hash output, upon which the second CAM output is based, is a new hash collision. Therefore, in a step 532 it is determined whether the stored search value in the returned search result is the same as the original input search value. If so, in a step 534 the process 500 is essentially finished for the present input search value, and the search engine 400 can carry on with whatever further operations on the search result are desired.

If the stored search value in the returned search result is not the same as the original input search value, in a step 536 the first CAM unit 424a can be programmed with the input search value and the second conflicts region 414c can be programmed with an associate value for this (all under direction of the programming unit 430). The process 500 is essentially finished for the present input search value.

If there were no CAM outputs (steps 524, 528), in a step 538 the second hash output is used to access the memory 414 and the controller 412 will receive a search result stored in the base region 414a.

In a step 540 the search result is examined to see if it contains any value. If so, there again still remains a possibility that the second hash output is a new hash collision, and in a step 542 the returned search result is compared with the original input search value.

In a step 544 it is determined whether these values are the same. If this is also so, in a step 546 the process 500 is essentially finished for the present input search value, and the search engine 400 can carry on with whatever further operations on the search result are desired.

In contrast, if the compared values are different, in a step 548 the second CAM unit 424b can be programmed with the first hash output and the first conflicts region 414b can be programmed with a search result (stored search value and associate value, all under direction of the programming unit 430). The process 500 is then essentially finished for the present input search value.

Lastly, if in step 540 it is determined that the search result is empty, in a step 550 a search result based on the input search value and an associate value for it may be programmed into the base region 414a of the memory 414, under direction of the programming unit 430. The process 500 is then essentially finished for the present input search value.

With reference now to FIG. 6-7 and TABLE 2, an example is presented. By definition here the input search values are six-digits long and the associate values are two-digits long (base

10 numbers are also used here to facilitate understanding). The search data bus **416** and the path **428a** are six-digit buses, the path **428b** is a four-digit bus, and the paths **428c-e** are all two-digit buses. The address bus **418** is a three-digit bus because it must carry the two-digit content of the second hash output when there are no hash collisions in the H-CAM **410**, and also carry three
5 digits otherwise (discussed presently).

Accordingly, the format 000:000000,00 can be used to reference the memory **414**, wherein the three digits to the left of the colon represent an address index value, the six digits between the colon and comma represent a stored search value, and the two digits to the right of the comma represent an associate value.

10 The contents of the first CAM unit **424a** can be represented using the format 000:000000, wherein the three digits to the left of the colon represent an address index value and the six digits to the right of the colon represent a comparison value equaling an input search value known to produce a collision in the first hash unit **422a**. That is, the 6-digit comparison value is what is compared for a match and the 3-digit address value is associatively "stored" with it (typically in CAM, where this address is simply based on the storage location of the comparison value), to be
15 used to avoid the hash collision by accessing an alternate location in the memory **414** (i.e., in the second conflicts region **414c**).

The contents of the second CAM unit **424b** can be represented using the format 000:0000, wherein the three digits to the left of the colon represent an address value and the four
20 digits to the right of the colon represent a comparison value for a hash output known to produce a collision in the second hash unit **422b**. That is, the 4-digit comparison value is what is compared for a match here and the 3-digit address value is associatively stored with it, to be used to avoid the hash collision here by accessing an alternate location in the memory **414** (i.e., in the first conflicts region **414b**).

25 Initially (step **510**), the CAM units **424a-b** and the memory **414** have nothing stored in them. Typically all the values will be zero, but other conventions can be used. This state is represented in row 1 of TABLE 2.

A first search value is then entered, say, 324238 (step **512**). The first hash unit **422a** generates a first hash output, say, 7783 (step **514**). With nothing stored yet in the first CAM unit
30 **424a**, it will not provide any first CAM output (step **516**). The second hash unit **422b** will generate a second hash result, say, 63 (step **518**). With nothing stored yet in the second CAM

unit **424b**, it will not provide any second CAM output (step **520**). The logic unit **426** therefore simply picks the second hash output, 63 (step **522**). Using this address value, the memory **414** provides the content of address 63, which is nothing so far (step **538**). This state is represented in row 2 of TABLE 2.

5 The controller **412** will determine that the first search value (324238) has no entry yet in the memory **414** (step **540**). One is therefore created in the base region **414a** using the search value and an associate value for it, say, 17. This entry is programmed into the memory **414** at address 63 (step **550**), since that is what the hash units **422a-b** here map the search value of 324238 to, and address 63 is presently available. This state is represented in row 3 of TABLE 2.

10 A second search value is entered, say, 578901. The first hash unit **422a** will generate a first hash result, say, 6311. The second hash unit **422b** will generate a second hash result, say, 63 again. With nothing stored yet in the CAM units **424a-b**, they do not create any CAM outputs, and the logic unit **426** picks the second hash output, 63. Using this address value, the memory **414** provides the content of address 63, which is 324238, 17. This state is represented in row 4 of TABLE 2.

15 The controller **212** now determines that: the search value (578901) has a corresponding entry in the memory **414**; a hash collision has occurred because the search field in the search result is different than the input search value used (324238≠578901)(step **542**); and that this is a first level hash collision. An entry in the second CAM unit **424b** is therefore created (step **548**) using the first hash output (6311) and an available address for an entry in a conflicts region (here the first conflicts region **414b**) of the memory **414**, say, address 200. Furthermore, a new entry is also created in the memory **414**, at address 100 in the first conflicts region **414b** based on the search value (578901) and an associate value provided for it, say, 23. This state is represented in row 5 of TABLE 2.

25 A third search value is entered, say, 322413. The first hash unit **422a** will generate a first hash output, say, 6311 again, which forces the second hash unit **422b** to generate 63 as the second hash output. The first CAM unit **424a** has no match for 322413, so there is no first CAM output. However, the second CAM unit **424b** has a match for 6311, so it provides the associatively stored address index 100 as the second CAM output (step **520**). The logic unit **426**
30 evaluates its inputs and then picks the second CAM output, 100. Using this as the address value, the memory **414** provides the content of address 100, which is 578901, 23.

The controller **412** now determines that: the search value (322413) has a corresponding entry in the memory **414**; a hash collision has occurred because the search field in the search result is different than the input search value used ($324238 \neq 322413$); this is not a first level hash collision; and that this is a second level hash collision. An entry in the first CAM unit **424a** is therefore created (step **536**) using the search value (322413) and an available address for an entry in the conflicts region (here the second conflicts region **414c**) of the memory **414**, say, address 200. Furthermore, a new entry is also created in the memory **414**, at address 200 in the second conflicts region **314c** and based on the search value (322413) and an associate value provided for it, say, 86. No search value is required in the second conflicts region **414c** the memory **414** because the first CAM unit **424a** already contains it and it will always map uniquely (a dash represents this in TABLE 2). This state is represented in row 7 of TABLE 2.

A fourth search value is entered, say, 578901 again. The first hash unit **422a** generates a first hash output of 6311, which forces the second hash unit **422b** to generate 63 as the second hash output. The first CAM unit **424a** has no a match for 578901, so there is no first CAM output. The second CAM unit **424b** has a match for 6311, so the second CAM output is the associatively stored address index 100. The logic unit **426** evaluates its inputs and then picks the second CAM result, 100. Using this address value, the memory **414** provides the content of address 100, which is 578901, 23.

The controller **412** now determines that: the search value (578901) has a corresponding entry in the memory **414**; and a new hash collision has not occurred because the search field in the search result is the same as the input search value used ($578901 = 578901$). No learning is therefore necessary and the contents of the associate value field, 23, can be used by the search engine **400**. This state is represented in row 8 of TABLE 2.

A fifth search value is entered, say, 322413 again. The first hash unit **422a** will generate a first hash output of 6311, which forces the second hash unit **422b** to generate 63 as the second hash output. The first CAM unit **424a** has a match for 322413, so the first CAM output is the associatively stored address index 200. The second CAM unit **424b** also has a match for 6311, so the second CAM output is the associatively stored address index 100. The logic unit **426** evaluates its inputs and then picks the first CAM result, 200 (since a first CAM output gets highest priority). Using this as the address value, the memory **414** provides the content of address 200, which is -, 86 (recall that there are no stored search values in the highest order

conflicts region).

The controller **412** now determines that: the search value (322413) has a corresponding entry in the memory **414**; and a new hash collision has not occurred because the search field in the result is the same as the input search value used (322413=322413). No learning is therefore necessary and the contents of the associate value field, 86, can be used by the search engine **400**. This state is represented in row 9 of TABLE 2.

Finally, a sixth search value is entered, say, 324238 again. The first hash unit **422a** will generate a first hash output of 7783 again, which forces the second hash unit **422b** to generate 63 again as the second hash output. The first CAM unit **424a** has no match for 324238, so there is no first CAM output. The second CAM unit **424b** has no match for 7783, so there also is no second CAM output. The logic unit **426** evaluates its inputs and then picks the second hash output, 63. Using this as the address value, the memory **414** provides the content of address 63, which is 324238, 17.

The controller **412** now determines that: the search value (324238) has a corresponding entry in the memory **414**; and a new hash collision has not occurred because the search field in the search result is the same as the input search value used (324238=324238). No learning is therefore necessary and the contents of the associate value field, 17, can be used by the search engine **400**. This state is represented in row 10 of TABLE 2.

A H-CAM INCLUDING A HASH POINTER AND A SEARCH ENGINE EMPLOYING IT:

FIG. 8 is a block diagram depicting a search engine **600** according to another preferred embodiment of the present invention. The particularly novel aspect here is a single level H-CAM **610** which employs a variation of the hash pointer (FIG. 3) to perform internal comparisons for detecting hash collisions. The H-CAM **610** works with a controller **612** and a memory **614**. These may be substantially conventional, but are both different in some regards from the controller **212** and memory **214** (FIG. 4). In this embodiment the controller **612** does not need to receive any stored search values back for performing comparisons against original input search values (reducing its pin-count and footprint), and the memory **614** does not need to store the search values (reducing its size).

As shown in FIG. 8, the architecture of the H-CAM **610** is similar in some respects to the

H-CAM 210 of FIG. 4. The H-CAM 610 includes a hash unit 622, a CAM unit 624, a first logic unit 640, and a comparison section 652. The salient difference is the comparison section 652, which includes a search data storage 654, a comparator 656, and a second logic unit 658.

With reference now also to FIG. 9a-b, these depict possible internal architectures for the first logic unit 640 and the search data storage 654. FIG. 9a is a block diagram depicting one architecture for the first logic unit 640, wherein it receives the hash output, the CAM output, and also a CAM hit signal (the mere presence of a CAM output denotes a hit, of course, but using a separate signal provides some speed and logic design benefits). The first logic unit 640 connects to the search data storage 654 and provides it with a hash address based on the hash output.

FIG. 9b is a block diagram depicting an architecture for the search data storage 654, wherein it is broken into two memory blocks: a hash pointer memory 660 and a search data memory 662. Conceptually the hash pointer memory 660 works much like the hash pointer unit 110 discussed with reference to FIG. 3. The hash pointer memory 660 here receives the hash address and provides a pointer value. It thus has a size based on the number of ways of set-associativity used. The pointer value is then used as an address input to the search data memory 662, which here contains the stored search values that are used to compare with the original input search values to detect hash collisions. The hash pointer memory 660 typically will be much deeper ($m > n$ in FIG. 9b) than the search data memory 662, because its purpose is to increase the utilization of the search data memory 662. Most of the hash pointer memory 660 will be empty but that is acceptable, since it is used to reduce the collision probability in the search data memory 662. With a proper memory depth ratio between the hash pointer memory 660 and the search data memory 662, the amount of the search data memory 662 required can be reduced.

As can be seen in FIG. 9a, the comparison section 652 provides the pointer value to the first logic unit 640 and it is what is ultimately used in this embodiment to access the base region of the memory 614. The second logic unit 658 passes the pointer value of a search to the first logic unit 640 (the mere presence of a value denotes a hit but using a separate signal, as is depicted, provides some speed and design benefits). Thus, if 4-way set-associativity is used the first logic unit 640 can provide four hash addresses to the comparison section 652 and it will provide up to one pointer value back. The controller 612 can then also be signaled, via a hit line 664, that the search result on the result bus 620 is a valid one.

With reference briefly again to TABLE 1, it also summarizes the difference between the

prior art hash-based and CAM-based approaches and a simple embodiment of the invention depicted in FIG. 8, and FIG. 9a-b. The specifications are essentially the same as for the prior art examples, except that a size of 128 x 100K is chosen for the CAM unit 624 based on an assumption that not more than 10% of the total database size need be devoted to handling hash collisions. As can be seen, the H-CAM 610 has the same speed as the prior art CAM-based approach. It has roughly the same total bus pin-count at the controller 612 as the prior art CAM-based approach (the hit line 664 may require as little as a single-pin) and it has a substantially lower pin-count than the prior art hash-based approach. This item bears some further comment. In the H-CAM 610 a result bus 620 can be as narrow as 32 bits wide, since it carries only the associate value. In contrast, the result busses 20, 220 have to be 160 bits wide to carry both the stored search value and associate value.

In TABLE 1 the H-CAM 610 is shown having roughly the same cost and power consumption as the prior art hash-based approach, but this is somewhat misleading. The H-CAM 610 here employs only 2-way set-associativity. If a higher degree of multi-way set-associativity is used additional cost and power savings can be had, providing the present invention with even greater advantage over the hash-based and CAM-based prior art.

For an example lets now consider a 4-way set-associative case wherein 128-bit wide search values are used to search a database of one million entries. The memory 614 needs to be 1M x 32 in size, to handle only the one million desired entries (100% utilization is possible because using the pointer values are used). Furthermore, since a high percentage of hash collisions can be handled by the use of multi-way set-associativity, the CAM unit 624 can be small, say, only 32K x 128 in size. It should particularly be noted that increasing the degree of multi-way set-associativity in this manner permits the CAM unit 624 to become decreasingly small, almost trivially small. This is very beneficial if actual content associative type memory is used (typically necessary where high speed is desired). Such true CAM requires more die area and consumes more power than most other types of memory, and usually is much more expensive. Reducing the amount of CAM used is therefore desirable.

Continuing with the example, the hash pointer memory 660 needs to be 4M x 20 and the search data memory 662 needs to be 1M x 128. The hash addresses need to be 22-bits wide, to address the four million locations in the hash pointer memory 660. The pointer values are 20-bits wide, since they only need to address the entries (associate content) in the memory 614 and to

also address the one million entries (stored search values) in the search data memory 662. The total memory required in the example above is equivalent to $4M \times 61$ ($1M \times 32 + 32K \times 128 + 4M \times 20 + 1M \times 128$), which is a marked reduction from the $4M \times 160$ a conventional 4-way set associative approach would require.

5 There are many different ways to implement the comparison section 652, and the architecture of these (and the hash algorithm) play an important role in overall efficiency. If designed correctly, this approach is able to reduce the total amount of memory (CAM and other) required. As can be seen in TABLE 1, this can produce a substantial cost savings. Basically, two things can be improved. The first is the handling of hash collisions by using a multi-way set
10 associative hash, and the other is to reduce the depth of the search data storage 654, because the search values are usually very wide.

A H-RAM AND A SEARCH ENGINE EMPLOYING IT:

15 FIG. 10 is a block diagram depicting a search engine 700 according to another preferred embodiment of the present invention. The particularly novel aspect here is a H-RAM 710 which also employs a variation of the hash pointer (FIG. 3) to perform internal comparisons for detecting hash collisions. The H-RAM 710 works with a controller 712 and a memory 714. As was the case for the H-CAM 610 of FIG. 8, these may be substantially conventional. The
20 controller 712 here also does not need to receive any stored search values back for performing comparisons (reducing its pin-count and footprint), and the memory 714 here also does not need to store the search values (reducing its size).

As shown in FIG. 10, the architecture of the H-RAM 710 is notable in that it does not employ a CAM. The underlying rationale here is that if a sufficiently high degree of multi-way
25 set associativity is provided, CAM is not needed to address hash collisions.

The H-RAM 710 includes a hash unit 722, a first logic unit 740, and a comparison section 752. The comparison section 752 includes a search data storage 754, a comparator 756, and a second logic unit 758.

30 With reference now also to FIG. 11a-b, FIG. 11a is a block diagram depicting one possible internal architecture for the first logic unit 740 and FIG. 11b is a block diagram depicting one possible internal architecture for the search data storage 754. Turning first to FIG.

11a, the first logic unit **740** here is simpler than the first logic unit **640** and in FIG. 9a. It converts the (n-bit) hash output into (m-bit) hash addresses, and it performs any desired translation of the (n-bit) indirect hash pointer into the address value for use with the memory **714**.

As can be seen by comparing FIG. 11b with FIG. 9b, the search data storage **754** and the search data storage **654**, and for that matter the comparison section **752** and the comparison section **652**, may be essentially the same.

A MULTI-LEVEL H-CAM WITH ROUTING LOGICS, AND A SEARCH ENGINE EMPLOYING IT:

FIG. 12 is a block diagram depicting a search engine **800** according to another preferred embodiment of the present invention, an embodiment incorporating many of the principals so far discussed into a H-CAM **810**. A controller **812** and memory **814** (potentially the same as those used in the embodiments of FIG. 8 and 10) are present again and key points to note are in the architecture of the H-CAM **810**.

The H-CAM **810** here includes hash blocks **862**, a CAM block **864**, a comparison section **866**, and a main logic unit **868**. The hash blocks **862** each include a paired input logic sub-unit **870** and hash unit **872**. The CAM block **864** includes an input logic sub-unit **874**, a number of CAM units **876** (typically equal in number to the hash blocks **862**), and an output logic sub-unit **878**. The comparison section **866** includes a search data storage **880**, a comparator **882**, and an output logic sub-unit **884**.

The input logic sub-units **870** of the hash blocks **862** route data off of a common input path **886** into the individual hash block **862** or onward to the CAM block **864**, or route the output data from a prior hash block **862** in similar manner. The input logic sub-unit **874** of the CAM block **864** routes data into the CAM units **876**, and the output logic sub-unit **878** combines and routes data from the CAM units **876**. The output logic sub-unit **884** of the comparison section **866** combines and routes data from the comparison section **866**. The data received from the last hash block **862**, the output logic sub-unit **878** of the CAM block **864**, and the output logic sub-unit **884** of the comparison section **866** is used by the main logic unit **868**, which outputs information to the comparison section **866**, to the memory **814**, and to the controller **812**, via a hit line **888**.

The H-CAM 810 can be user configured to work as a 4-level unit, much in the manner described for the H-CAM 310 in FIG. 5; or it can be configured to work as a 2-level unit, much in the manner described for the H-CAM 410 in FIG. 6; or it can be configured to work as a 1-level unit, much in the manner described for the H-CAM 210 in FIG. 4. In particular, it can be configured to work as a 1-level unit similar to the H-CAM 610 and which operates on very large data. The hash blocks 862 can be configured to each handle part of a wide input on the common input path 886, in parallel, or they can be operated in serial fashion as has essentially been described elsewhere herein. The CAM block 864 can handle either case, with its input logic sub-unit 874 and its output logic sub-unit 878 configurable to use the CAM units 876 as one effective large CAM or as smaller amounts of CAM. The comparison section 866 and its output logic sub-unit 884 are similarly configurable to work with the configuration of the other components.

CASCADING IN A SEARCH ENGINE, SHOWN WITH H-CAMS:

FIG. 13 is a block diagram depicting a search engine 900 according to another preferred embodiment of the present invention, an embodiment employing the ability to scale for large needs by cascading multiple H-CAMs 910 to form a bigger overall system. If the internal architecture of the H-CAMs 910 generally resembles that of the H-CAMs 210, 310, 410, and an expansion bus 922 is added, a controller 912, memory 914, search data bus 916, address bus 918, and result bus 920 may be used which are essentially similar to those used there with. Alternately, if the internal architecture of the H-CAMs 910 resembles that of the H-CAMs 610, 810, and the expansion bus 922 is added, the controller 912, memory 914, result bus 920, and a hit line 924 may be used which are essentially similar to those used there with. The memory 914 may accordingly store the search data values and the associate content values or just the associate content values.

These architectures combines all the benefits of hash algorithm and CAM to provide an efficient and cost effective way to support searching large databases. The search engine 900 can support very large database searching without using large amounts of memory to accommodate possible hash collisions, yet it can achieve the same search performance as prior art CAM-based approaches without the attendant high cost and power consumption. Although this approach cannot guarantee a specific capacity to avoid hash collisions, similar to prior art hash-based

approaches, with the proper hash algorithm and number of ways of set-associativity this variation of the invention can nonetheless achieve very high utilization. This coincides well with a key goal of the present invention, to increase the degree or number of ways of set-associativity without increasing the memory speed or the bandwidth requirement.

5 A cascaded approach makes use of the power of distributed processing. If each H-CAM supports an n-way-set-associative hash, the entire system would be able to support (n x number of H-CAM)-way-set associative hash. Multiple H-CAMs, such as the H-CAM 610 shown in FIG. 8, can be cascaded together, such as through the expansion bus 922 as shown in FIG. 13. If n=2 and five H-CAMs are cascaded together, the system can employ a 10-way-set-associative
10 hash and the hash pointer memory and the search data memory within each H-CAM need to only support two reads per input search value, instead of the ten that would be needed in a typical prior art 10-way-set-associative hash approach.

TABLE 3 provides a comparison of two different implementations of a one million entry 8-way-set-associative hash-based system. One uses a typical prior art hash-based approach, while the other uses the H-CAM with the indirect hash pointer and a cascading mechanism. If each H-CAM supports a 2-way-set-associative hash, four are needed to implement this. Each H-CAM will have 256K entries in the search data memory and 512K pointers in the hash pointer memory. Further assuming each search data entry is 128-bits wide, it can be seen that the inventive approach uses only about one-sixth of the memory and requires only one-fourth of the
15 memory bandwidth.

CASCADING IN A SEARCH ENGINE, SHOWN WITH H-RAMS:

FIG. 14 is a block diagram depicting a search engine 1000 according to another preferred
25 embodiment of the present invention, an embodiment employing the ability to scale for large needs by cascading multiple H-RAMs 1010. In structure the search engine 1000 here much resembles the search engine 900 of FIG. 13, a controller 1012, memory 1014, search data bus 1016, address bus 1018, and result bus 1020 are provided, and an expansion bus 1022 is used to interconnect the H-RAMs 1010.

30 CASCADING IN A SEARCH ENGINE, SHOWN WITH CAMS, H-CAMS AND H-RAMS IN

COMBINATION:

FIG. 15 is a block diagram depicting a search engine 1100 according to another preferred embodiment of the present invention, an embodiment employing the ability to scale for large needs by cascading multiple of CAMs 1110a, H-CAMs 1110b, and H-RAMs 1110c, with a controller 1112, memory 1114, search data bus 1116, address bus 1118, and result bus 1120 and all interconnected by an expansion bus 1122. The search engine 1100 thus is particularly powerful and configurable to various applications, including searching very large and highly specialized databases.

SUMMARY:

A lot has been presented herein. The prior art hash approach was discussed, including multi-way set-associativity (see e.g., FIG. 1 (background art)), and the prior art CAM approach was also discussed (see e.g., FIG. 2 (background art)). A novel hash pointer approach was then discussed (see e.g., FIG. 3), including how it provides the benefits of multi-way set-associativity without the poor memory utilization that is characteristic of the prior art.

A novel H-CAM approach was next discussed, with coverage of it including how it provides many of the benefits of the respective hash and CAM approaches, but without many of the respective disadvantages. FIG. 4 depicted a single level H-CAM variation, and FIG. 5 depicted a multi level H-CAM variation. Both of these examples use 1-way set-associativity, for simplicity, but using multi-way set-associativity is a straight forward extension. FIG. 6 covered programming, which is essentially also straight forward and possible with generally conventional techniques, and FIG. 6 and the flow chart of FIG. 7 were used in a detailed example.

A novel "local" comparison approach for moving the task of comparison into the H-CAM was discussed (see e.g., FIG. 8 and 9a-b), employing a variation of the hash pointer approach (FIG. 3). As is the case with just the hash pointer approach, this local comparison approach provides particular benefit when used with multi-way set-associativity.

A novel H-RAM approach was then discussed, one which uses a variation of the local comparison approach (see e.g., FIG. 10 and 11a-b). Enhancing the multi level H-CAM by using internal routing logics was also discussed, with optional local comparison included (see e.g.,

FIG. 12). Finally, cascading multiple H-CAMs, cascading multiple H-RAMs, and cascading multiple CAMs, H-CAMs, and H-RAMs together in combination were shown (FIG. 13-15).

Accordingly, while various embodiments have been described above, it should be understood that they have been presented by way of example only, and not limitation. Thus, the

- 5 breadth and scope of the invention should not be limited by any of the above described exemplary embodiments, but should be defined only in accordance with the following claims and their equivalents.

09927599-080901
T06080-66522660

INDUSTRIAL APPLICABILITY

The present search engines 100, 200, 300, 400, 600, 700, 800, 900, 1000, 1100, and the individual H-CAMs 210, 310, 410, 610, and 810 and the individual H-RAM 710 are well suited for application in the searching of databases of associate content. The search engines, H-CAMs, and H-RAM can search large databases, particularly when variations such as the configurable embodiment depicted in FIG. 12 or the cascade able embodiments depicted in FIG. 13-15 are used.

The invention can substantially reduce the memory needed for database storage. The embodiment depicted in FIG. 3 shows how the invention permits memory size to not be based on the degree of multi-way set-associativity used. The embodiments depicted in FIG. 4-6 show how database memory may also be reduced by not storing instances of the search values in a highest level conflicts region in the main memory, and the embodiments depicted in FIG. 8-12 show how database memory may be reduced to store only associate content, and then with a very high degree of utilization.

The invention can also substantially reduce the time needed for conducting database searches. While the embodiments depicted in FIG. 3-6 show that the invention can at least equal the search speed of hash-based prior art approaches, the more sophisticated embodiments depicted in FIG. 8-11 show how the invention can better the search speed of hash-based prior art approaches and approach the speed of CAM-based prior art approaches.

The invention can also substantially reduce the overall search engine and database storage system cost. This is achieved directly by increasing memory utilization, and thus reducing the need for memory. Cost reduction is also further achieved indirectly, by reducing bus widths, pin counts, circuit foot print, power consumption (and dissipating heat), etc.

For the above, and other, reasons, it is expected that the present invention will have widespread industrial applicability. Therefore, it is expected that the commercial utility of the present invention will be extensive and long lasting.